WEEK 1

Day 1

Q1-SET MATRIX ZEROS

```
class Solution {
public:
   void setZeroes(vector<vector<int>>& matrix) {
      vector<pair<int,int>> v;
      int rows = matrix.size();
      int cols = matrix[0].size();
      for(int i = 0; i < rows; i++)
         for(int j = 0; j < cols; j++)
            if(matrix[i][j] == 0)
               v.push_back(make_pair(i,j));
      for(int i = 0; i < v.size(); i++){
         for(int j = 0; j < cols; j++){
            matrix[v[i].first][j] = 0;
            cout<<v[i].first;
         }
         for(int j = 0; j < rows; j++)
            matrix[j][v[i].second] = 0;
      }
   }
};
```

Q2 – Pascal's Triangle

```
class Solution {
public:
```

```
    vector<vector<int>> generate(int numRows) {
        vector<vector<int>> ans;
        for(int i=1;i<=numRows;i++){
            vector<int> a;
            for(int j=1;j<=i;j++){
                if(j==1||j==i)a.push_back(1);
                else{
                    int t=ans[i-2][j-2]+ans[i-2][j-1];
                    a.push_back(t);
                }
            }
            ans.push_back(a);
        }
        return ans;

    }
};
```

Q3- NEXT PERMUTATION

```
public:
    void nextPermutation(vector<int>& nums) {
        int n = nums.size()-1;
int inflectionPoint=0;

        for(int i=n; i>0; i--){
            if(nums[i]>nums[i-1]){
               inflectionPoint=i;
                break;
            }
```

```
        }


        if(inflectionPoint==0){
           sort(nums.begin(),nums.end());
        }
        else{
        int toSwap= nums[inflectionPoint-1];
            for(int j=inflectionPoint; j<=n; j++){
                if(nums[j]-toSwap>0){
                  int temp=nums[j];
                  nums[j]=nums[inflectionPoint-1];
                  nums[inflectionPoint-1]=temp;
                }
            }
             sort(nums.begin()+inflectionPoint, nums.end());
}
    }
};
```

Q4-Maximum Subarray

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
         int n = nums.size();
         int curr_sum = 0;
        int largest_sum = INT_MIN;
        for(int i = 0; i < n; i++)
        {
           curr_sum += nums[i];
```

```
        largest_sum = max(largest_sum, curr_sum);
        if(curr_sum < 0)
        {
            curr_sum = 0;
        }
      }
       return largest_sum;
   }
  };
```

Q5 – Sort an Array of 0’s 1’s 2’s

```
class Solution {
public:
   void sortColors(vector<int>& nums) {
int lo = 0;
       int hi = nums.size() - 1;
       int mid = 0;

       while (mid <= hi) {
          switch (nums[mid]) {
          case 0:
             swap(nums[lo++], nums[mid++]);
             break;
          case 1:
             mid++;
             break;
          case 2:
             swap(nums[mid], nums[hi--]);
```

```
            break;
        }
    }
}
};
```

Q6- Stock Buy and Sell

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int n = prices.size();
        int maxp = 0, minp = INT_MAX;
        for(int i = 0; i < n; i++){
            minp = min(minp, prices[i]);
            maxp = max(maxp, prices[i] - minp);
        }
        return maxp;
    }
};
```

DAY 2

Q7-Rotate Matrix

```
class Solution {
public:
   void rotate(vector<vector<int>>& matrix) {
       int m = matrix.size();
       for(int i=0;i<m;i++){
           for(int j=0;j<i;j++){
               swap(matrix[i][j],matrix[j][i]);
           }
       }
      int n = matrix[0].size();
      for(int i=0;i<m;i++){
           for(int j=0;j<n/2;j++){
               swap(matrix[i][j],matrix[i][n-1-j]);
           }
       }
   }
};
```

Q9 – Merge Overlapping Subintervals

```
class Solution {
public:
   void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
       vector<int>ans;
       int i=0,j=0;
       while (i<m&&j<n){
```

```
        if(nums1[i]<nums2[j]){
            ans.push_back(nums1[i]);
            i++;
        }
        else{
                ans.push_back(nums2[j]);
            j++;
        }
      }
      while(i<m){
        ans.push_back(nums1[i]);
        i++;
      }
      while(j<n){
        ans.push_back(nums2[j]);
        j++;
      }
      nums1=ans;
     }
};
```

Q10 – Find the duplicate in an array of N+1 integers

```
class Solution {
public:
   int findDuplicate(vector<int>& nums) {
```

```
        int low = 1, high = nums.size() - 1, cnt;
       while(low <=  high)
        {
            int mid = low + (high - low) / 2;
            cnt = 0;
             cnt number less than equal to mid
            for(int n : nums)
            {
                if(n <= mid)
                    ++cnt;
            }
             binary search on left
            if(cnt <= mid)
                low = mid + 1;
            else
             binary search on right
                high = mid - 1;
          }
        return low;
    }
};
```

Q11- Repeated And Missing Number

```
vector<int> Solution::repeatedNumber(const vector<int> &A) {
    int a, b;
    vector<bool> temp(A.size(), false);
    for(int i=0;i<A.size();i++){
```

```
        if(temp[A[i]]) a = A[i];
        temp[A[i]] = true;
    }
    for(int i=1; i<=A.size();i++){
        if(temp[i]==false) b = i;
    }
    vector<int> ans;
    ans.push_back(a); ans.push_back(b);
    return ans;}
```

Q 12- Inversion Of Array

```
#include<bits/stdc++.h>
using namespace std;

int merge(int arr[],int temp[],int left,int mid,int right)
{
    int inv_count=0;
    int i = left;
    int j = mid;
    int k = left;
    while((i <= mid-1) && (j <= right)){
        if(arr[i] <= arr[j]){
            temp[k++] = arr[i++];
        }
        else
        {
            temp[k++] = arr[j++];
            inv_count = inv_count + (mid - i);
        }
```

```
    }

    while(i <= mid - 1)
        temp[k++] = arr[i++];

    while(j <= right)
        temp[k++] = arr[j++];

    for(i = left ; i <= right ; i++)
        arr[i] = temp[i];

    return inv_count;
}

int merge_Sort(int arr[],int temp[],int left,int right)
{
    int mid,inv_count = 0;
    if(right > left)
    {
        mid = (left + right)/2;

        inv_count += merge_Sort(arr,temp,left,mid);
        inv_count += merge_Sort(arr,temp,mid+1,right);

        inv_count += merge(arr,temp,left,mid+1,right);
    }
    return inv_count;
}
```

```
int main()
{
    int arr[]={5,3,2,1,4};
    int n=5;
    int temp[n];
    int ans = merge_Sort(arr,temp,0,n-1);
    cout<<"The total inversions are "<<ans<<endl;



    return 0;
}
```

DAY 3

Q13-Search in a 2d Matrix

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        bool a = false;
       int i=0,j=0, ad;
      while(i<matrix.size() && j<matrix[0].size()){
            if(matrix[i][j]==target){
                a=true;
                break;
            }
            if(i+1<matrix.size()){
            if(matrix[i+1][j]>target) j++;
            else if(matrix[i+1][j]<=target) i++;
            }
            else if(i+1 == matrix.size())j++;
          }
        return a;
    }
};
```

Q14 –Pow(X,n)

```
class Solution {
public:
    double myPow(double x, int n) {
```

```
        double ans = 1.0;
        long long power_of_x = abs(n);

        while(power_of_x>0)
        {

            if(power_of_x%2==1)
            {
                ans = ans*x;
                power_of_x = power_of_x - 1;
            }

            else if(power_of_x%2==0)
            {
                x = x*x;
                power_of_x = power_of_x / 2;
            }
        }

        if(n<0)
            return (double)1/(double)ans;

        return (double)ans;
    }
};
```

Q15-Majority Element (N/2)

```
class Solution {
public:
int majorityElement(vector<int>& nums) {
```

```
    int count=0;
        int majNumber;
        int size = nums.size();
        for (int i=0;i<size;i++){
            if (count==0){
                majNumber = nums[i];
            }
            if (nums[i]==majNumber){
                count++;
            }else{
                count--;
            }

        }
        return majNumber;
    }
    }
};
```

Q 16-Majority Element (N/3)

```
class Solution {
public:
    vector<int> majorityElement(vector<int>& nums) {
   int majElem1=0,majElem2=0,count1=0,count2=0,n=nums.size();
        vector<int> ans;
        for(int i=0;i<n;i++){
            if(majElem1==nums[i]){
                count1++;
```

```
        }else if(majElem2==nums[i]){
            count2++;
        }else if(count1==0){
            majElem1=nums[i];
            count1++;
        }else if(count2==0){
            majElem2=nums[i];
            count2++;
        }else{
            count1--;
            count2--;
        }
    }
    count1=count2=0;
    for(int i=0;i<n;i++){
        if(nums[i]==majElem1){
            count1++;
        }else if(nums[i]==majElem2){
            count2++;
        }
    }
    if(count1>n/3) ans.push_back(majElem1);
    if(count2>n/3) ans.push_back(majElem2);
    return ans;
}
};
```

Q 17-Grid Unique Paths

```
class Solution {
public:
```

```
  int uniquePaths(int m, int n) {
      int M = m+n-2;
      int R=n-1;
      double a = 1;

      for(int i=1;i<=R;i++){
         a=a*(M-R+i)/i;
      }
      return (int)a;
   }
};
```

Q 18-Reverse Pairs

```
class Solution {
public:
   int Merge(vector < int > & nums, int low, int mid, int high) {

  int total = 0;
  int j = mid + 1;
  for (int i = low; i <= mid; i++) {
    while (j <= high && nums[i] > (2*(long long)nums[j])) {
     j++;
    }
    total += (j - (mid + 1));
  }

  vector < int > t;
  int left = low, right = mid + 1;

  while (left <= mid && right <= high) {
```

```
    if (nums[left] <= nums[right]) {
      t.push_back(nums[left++]);
    } else {
      t.push_back(nums[right++]);
    }
  }

  while (left <= mid) {
    t.push_back(nums[left++]);
  }
  while (right <= high) {
    t.push_back(nums[right++]);
  }

  for (int i = low; i <= high; i++) {
    nums[i] = t[i - low];
  }
  return total;
}
int MergeSort(vector < int > & nums, int low, int high) {

  if (low >= high) return 0;
  int mid = (low + high) / 2;
  int inv = MergeSort(nums, low, mid);
  inv += MergeSort(nums, mid + 1, high);
  inv += Merge(nums, low, mid, high);
  return inv;
}
```

```
int reversePairs(vector < int > & nums) {
  return MergeSort(nums, 0, nums.size() - 1);
}


};
```

DAY 4

Q19 -2 SUM

```
class Solution {
public:
   vector<int> twoSum(vector<int>& nums, int target) {
        vector<int> a;
       for(int i=0;i<nums.size() ;i++){
        for(int j=i+1;j!=nums.size();j++){
           if(nums.at(i)+nums.at(j) == target){
               a.push_back(i);
               a.push_back(j);

        }
    }
   }
       return a;
   }
};
```

Q20-4 sum

```
class Solution {
public:
   vector<vector<int>> fourSum(vector<int>& nums, int target) {
       vector<vector<int>> res;

       if(nums.empty())
           return res;
```

```
int n = nums.size();
sort(nums.begin(),nums.end());

for(int i=0; i<n; i++)
{
   long long int target3 = target - nums[i];
  for(int j=i+1; j<n; j++)
   {
       long long int target2 = target3 - nums[j];
      int front = j+1;
       int back = n-1;
     while(front<back)
      {
          int two_sum = nums[front] + nums[back];
        if(two_sum < target2)
            front++;
          else if(two_sum > target2)
            back--;
        else
         {
            vector<int> quad(4,0);
            quad[0] = nums[i];
            quad[1] = nums[j];
            quad[2] = nums[front];
            quad[3] = nums[back];
            res.push_back(quad);
            while(front < back && nums[front] == quad[2])
               front++;
            while(front < back && nums[back] == quad[3])
```

```
                back--;
            }
        }
            while(j + 1 < n && nums[j + 1] == nums[j])
                j++;
        }
        while(i + 1 < n && nums[i + 1] == nums[i])
                i++;
    }
    return res;
  }
};
```

Q21 Longest Consequence Problem

```
class Solution {
public:
  int longestConsecutive(vector<int>& nums) {

     int n = nums.size();
  int streak = 1;
  int maxs = 0;

  if(n == 0)
    return 0;

  sort(nums.begin(), nums.end());

  for(int i = 1; i < n; i++){
```

```
        if(nums[i] - 1 == nums[i - 1])
        {

            streak++;
        }
        else if(nums[i] != nums[i - 1])
        {
            maxs = max(streak, maxs);
            streak = 1;
        }

    }

    return max(streak, maxs);

    }
};
```

Q 22 Largest Subarray with 0

```
class Solution{
    public:
    int maxLen(vector<int>&A, int n)
    {
        // Your code here
        unordered_map<int,int> mpp;
    int maxi = 0;
    int sum = 0;
    for(int i = 0;i<n;i++) {
```

```
        sum += A[i];
        if(sum == 0) {
            maxi = i + 1;
        }
        else {
            if(mpp.find(sum) != mpp.end()) {
                maxi = max(maxi, i - mpp[sum]);
            }
            else {
                mpp[sum] = i;
            }
        }
    }

    return maxi;
    }
};
```

Q 23-Count number of Subarray with given Xor K

```
int Solution::solve(vector<int> &A, int B) {
  unordered_map<int,int>visited;
    int cpx = 0;
    long long c=0;
    for(int i=0;i<A.size();i++){
        cpx^=A[i];
        if(cpx==B) c++;
        int h = cpx^B;
        if(visited.find(h)!=visited.end()){
            c=c+visited[h];
```

```
        }
        visited[cpx]++;
    }
    return c;
}
```

Q24-Longest Substring without repeating characters

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        int maxans = INT_MIN;
  for (int i = 0; i < s.length(); i++)
  {
    unordered_set < int > set;
    for (int j = i; j < s.length(); j++)
    {
      if (set.find(s[j]) != set.end())
      {
        maxans = max(maxans, j - i);
        break;
      }
      set.insert(s[j]);
    }
  }
  return maxans;
    }
};
```

DAY 5

Q25-

Reverse a Linked List

```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {

        ListNode* prev=NULL;
        ListNode* curr=head;
        ListNode* next=NULL;

        while(curr!=NULL){
            next=curr->next;
            curr->next=prev;
            prev=curr;
            curr=next;
        }
        return prev;
    }
};
```

Q 26-

Find the middle of linked list

```
class Solution {
public:
    ListNode* middleNode(ListNode* head) {
      if(head == NULL) return NULL;
        if(head->next == NULL) return head;
```

```
        ListNode*slow = head;
        ListNode*fast = head;
        while(fast && fast->next){
            slow = slow->next;
            fast = fast->next->next;
        }
        return slow;
    }
};
```

Q27- Merge Two Sorted Linked List

```
class Solution {

public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        if(l1 == NULL) return l2;
        if(l2 == NULL) return l1;
        if(l1->val > l2->val) std::swap(l1,l2);
        ListNode* res = l1;
while(l1 != NULL && l2 != NULL) {
 ListNode* temp = NULL;
while(l1 != NULL && l1->val <= l2->val) {
temp = l1;//storing last sorted node
                l1 = l1->next;
            }
            temp->next = l2;
            std::swap(l1,l2);
        }
```

```
return res;
   }
};
```

Q28-Remove N-th Node From Back of Linked List

```
class Solution {
public:
   ListNode* removeNthFromEnd(ListNode* head, int n) {
       ListNode *fast = head, *slow = head;
      for(int i=0; i<n; i++){
         fast = fast->next;
      }

      if(fast == NULL)
         return head->next;

      while(fast->next != NULL){
         slow = slow->next;
         fast = fast->next;
      }
     slow->next = slow->next->next;
     return head;
    }
};
```

Q29-ADD TWO NUMBER AS LINKED LIST

```
class Solution {
public:
   ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
```

```
vector<int> a;
ListNode * new_head = l1, *cur = l1;

while(l1 && l2)
{
   a.emplace_back(l1->val+l2->val);
   l1 = l1->next;
   l2=l2->next;
}
while(l1)
{
   a.emplace_back(l1->val);
   l1 = l1->next;
}
while(l2)
{
   a.emplace_back(l2->val);
   l2 = l2->next;
}

int carry = 0; int x, rem;
for(int i = 0; i<a.size(); i++)
{
   x =  (a[i]+carry);
    // cout << x <<" ";
   rem = x%10;
   carry = x/10;
   cur->val = rem;
   if(cur->next == NULL && (i<a.size()-1 || carry))
```

```
        {
            ListNode * temp = new ListNode();
            cur->next = temp;
        }
        cur = cur->next;
      }
      if( carry )
        {
            cur->val = carry;
        }
      return new_head;
   }
};
```

Q30-Delete a given Node when a node is given

```
class Solution {
public:
   void deleteNode(ListNode* node) {
    node->val = node->next->val;
   node->next = node->next->next;
   return;
   }
};
```

DAY 6

Q31-Find Intersection Point of Y Linked List

```
class Solution {
public:

   ListNode *getIntersectionNode(ListNode *headA, ListNode *headB) {
     ListNode* d1 = headA;
   ListNode* d2 = headB;

   while(d1 != d2) {
      d1 = d1 == NULL? headB:d1->next;
      d2 = d2 == NULL? headA:d2->next;
   }

   return d1;
   }
};
```

Q 32-Detect a Cycle in Linked List

```
class Solution {
public:
   bool hasCycle(ListNode *head) {
      if(head == NULL) return false;
   ListNode* fast = head;
   ListNode* slow = head;

   while(fast->next != NULL && fast->next->next != NULL) {
      fast = fast->next->next;
```

```
        slow = slow->next;
        if(fast == slow) return true;
    }
    return false;
    }
};
```

Q33-Reverse Nodes in K -Group

```
class Solution {
public:
    ListNode* reverseKGroup(ListNode* head, int k) {
        vector<int> v;
    ListNode *p= head;
    while(p!=NULL)
    {
        v.push_back(p->val);
        p=p->next;
    }
    int n=v.size();
    for(int i=0;i<=n-k;i+=k)
    {
        reverse(v.begin()+i,v.begin()+i+k);
    }
    ListNode* l1, *l2, *l3=NULL;
    int ok=1;
    for(int i=0;i<n;i++)
    {
        l1= new ListNode(v[i]);
        if(ok==1) l3=l1,l2=l1,ok=0;
```

```
        else l2->next=l1,l2=l2->next;
    }
    return l3;
    }
};
```

Q34-Check if a Linked List is Palindrome or not

```
class Solution {
public:
    bool isPalindrome(ListNode* head) {
        vector<int> arr;
    while(head != NULL) {
        arr.push_back(head->val);
        head = head->next;
    }
    for(int i=0;i<arr.size()/2;i++)
        if(arr[i] != arr[arr.size()-i-1]) return false;
    return true;
    }
};
```

Q35- Find The Starting point of Loop of Linked List

```
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
         if(head == NULL||head->next == NULL) return NULL;

    ListNode* fast = head;
     ListNode* slow = head;
```

```
    ListNode* entry = head;

   while(fast->next != NULL&&fast->next->next != NULL) {
      slow = slow->next;
      fast = fast->next->next;

      if(slow == fast) {
         while(slow != entry) {
            slow = slow->next;
            entry = entry->next;
         }
         return slow;
      }
   }
   return NULL;
   }
};
```

Q36-Flattening of Linked List

```
 Node* mergeTwoLists(Node* a, Node* b) {

   Node *temp = new Node(0);
   Node *res = temp;

   while(a != NULL && b != NULL) {
      if(a->data < b->data) {
         temp->bottom = a;
         temp = temp->bottom;
         a = a->bottom;
```

```
        }
        else {
            temp->bottom = b;
            temp = temp->bottom;
            b = b->bottom;
        }
    }

    if(a) temp->bottom = a;
    else temp->bottom = b;

    return res -> bottom;

}
Node *flatten(Node *root)
{
   if (root == NULL || root->next == NULL)
            return root;
       root->next = flatten(root->next);
       root = mergeTwoLists(root, root->next);
       return root;
}
```

DAY 7

Q37 – Rotate A Linked List

```
class Solution {
public:
   ListNode* rotateRight(ListNode* head, int k) {
       if(head == NULL||head->next == NULL||k == 0) return head;
   //calculating length
   ListNode* temp = head;
   int length = 1;
   while(temp->next != NULL) {
      ++length;
      temp = temp->next;
   }
   //link last node to first node
   temp->next = head;
   k = k%length; //when k is more than length of list
   int end = length-k; //to get end of the list
   while(end--) temp = temp->next;
   //breaking last node link and pointing to NULL
   head = temp->next;
   temp->next = NULL;
   return head;
   }
};
```

Q38-Clone a Linked List with random and next pointer

```
class Solution {
public:
```

```
  Node* copyRandomList(Node* head) {
if(!head) return head;

    Node *ptr = head;
    while(ptr) {
       Node* tmp = new Node(ptr->val);
       tmp->next = ptr->next;
       ptr->next = tmp;
       ptr = ptr->next->next;
    }
    ptr = head;
    while(ptr) {
       ptr->next->random = (ptr->random) ? ptr->random->next : NULL;
       ptr = ptr->next->next;
    }

    Node* dummy = new Node(-1);
    ptr = dummy;
    Node* curr = head;
    Node* nxt = head;

    while(nxt) {
       nxt = curr->next->next;
       ptr->next = curr->next;
       ptr = ptr->next;
```

```
            curr->next = nxt;
            curr = curr->next;
        }

        return dummy->next;
    }};
```

Q39- 3Sum

```
class Solution {
public:
    vector<vector<int>> threeSum(vector<int>& nums) {
         vector<vector<int>> result;
        int n = nums.size();
        sort(nums.begin(), nums.end());
        int index = 0;
        for(int index = 0; index<n; index++){
            if(index>0 && nums[index]==nums[index-1]){
                continue;
            }
            int si = index+1;int ei = n-1;
            while(si<ei){
                int sum = nums[index]+nums[si]+nums[ei];
                if(sum==0){
                    result.push_back(vector<int>{nums[index], nums[si], nums[ei]});
                    while(si<n-1 && nums[si]==nums[si+1]){
```

```
                si++;
            }
            while(ei>0 && nums[ei]==nums[ei-1]){
                ei--;
            }
            si++;ei--;
        }
        else if(sum<0){
            si++;
        }else{
            ei--;
        }
      }
    }

    return result;
  }
};
```

Q40-Trapping Rainwater

```
class Solution {
public:
   int trap(vector<int>& height) {

 int n = height.size(), water = 0;
      int left = 0, right = n-1, leftMax = 0, rightMax = 0;
```

```
        while(left < right) {
            if(height[left] <= height[right]) {
                if(leftMax > height[left]) water += (leftMax - height[left]);
                else leftMax = height[left];
                ++left;
            }
            else {
                if(rightMax > height[right]) water += (rightMax - height[right]);
                else rightMax = height[right];
                --right;
            }
        }


        return water;
    }
};
```

Q 41-

Remove Duplicate from Sorted Array

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
      int i=1;
int n = nums.size();
int count = 0;
int large = nums[n-1]; //largest element of array as array is sorted
while(i<n){
```

```
if(nums[i]==nums[i-1]){
nums[i-1]=large+1; //putting largest element in place of duplicate elements. We will sort the final array in the end to get desired array.
count++;
}
i++;
}
sort(nums.begin(), nums.end());
return n-count;

   }
};
```

Q42-Max Consecutive Ones

```
class Solution {
public:
   int findMaxConsecutiveOnes(vector<int>& nums) {
      int cnt =0;
int maxi=0;
for(int i=0;i<nums.size();i++){
if(nums[i]==1){
cnt++;
}else{
cnt=0;
}
maxi=max(maxi,cnt);
}
return maxi;
```

```
    }
};
```